Parallel Solutions of Multi-Stage Stochastic Linear Programs by Interior-Point Methods

Joseph Czyzyk
Robert Fourer
Sanjay Mehrotra

Technical Report 94-??

Department of Industrial Engineering and Management Sciences

Northwestern University, Evanston, Illinois 60208-3119

August 1994

This work has been supported in part by: National Science Foundation grants CCR-8810107 to Sanjay Mehrotra and DDM-8908818 to Robert Fourer; Office of Naval Research grant N00014-93-1-0317 to Sanjay Mehrotra and Robert Fourer; and a Department of Energy Computational Science Graduate Fellowship to Joseph Czyzyk.

1. Introduction

In this paper, we demonstrate how interior-point techniques and parallel programming methods can be used together to solve stochastic linear programs (SLP's). Linear programs (LP's) are often used to model complex interactions since they are relatively easy to understand and there are efficient methods for solving them. Just as with any model, the quality of the solution can be no better than the quality of the input data. This holds true for linear programs as well and poses a significant problem for many models. Consider, for example, a long-range planning problem. The values of the problem data (cost of commodities, demand, inventory, etc.) for the present time can probably be determined accurately. The information for future planning periods, however, is not readily available and may need to be forecast. Since the future is uncertain, making plans based on one predicted instance of the future may lead to suboptimal (even disastrous) results when the future events that occur are dramatically different from the predicted event. Taking this uncertainty into account is vital for wise planning. A stochastic linear program is a reformulation of the standard linear program that takes the uncertainty into account. These problems are typically large and, as a result, more difficult to solve.

The advent and development of parallel and supercomputers has made many difficult problems computationally feasible. By using our algorithm it is possible to partition the problem into subproblems of roughly equal size which can be computed on separate processors of a parallel machine. The CM-5 is an MIMD machine composed of many Sparc processors connected by control, data, and diagnostic networks. We use up to 128 processors to solve our problems.

Lustig, Mulvey and Carpenter [13] discussed the use of interior-point methods for solving stochastic linear programs. They experimented with the use of different splitting techniques to eliminate the set of dense columns that occurs in the constraint matrix. Czyzyk, Fourer, and Mehrotra [6] extended their work to show that the partial-splitting scheme may not always reduce the amount of work to solve the problem but that the use of an augmented system approach limits the growth in solution time to be linear in the number of scenarios included in the problem.

Others have implemented similar methods to solve stochastic linear programs on parallel machines. One method is to use Benders Decomposition (also called the L-Shaped method [16]) which decomposes the problem into a master LP and various smaller LP's. Birge [1] extended these ideas to the multi-stage case and devised the nested decomposition method. Birge, et al. [3] implemented this algorithm on a network of RS/6000 workstations. Another approach uses interior-point methods to solve the problem. Birge and Holmes [4] use the decomposition method of Birge and Qi [5] to solve the problems both with serial and parallel implementations. Their approach factors a system of equations using a variant of the Sherman-Morrison-Woodbury method. Although the method works well in most cases, they report numerical instability for some problems. Jessup, et al. [11] use the Birge-Qi procedure in an implementation tailored for the iPSC/860 and CM-5. de Silva and Abramson [7] decompose a system of equations for parallel factorization and use an indefinite solver of Vanderbei and Carpenter [15]. They solve a set of financial optimization problems on a 128-CPU Fujitsu AP1000.

The remainder of this paper discusses the formulation of stochastic linear programs and how their structure can be exploited in our method. Section 2 describes the method, and Section 3 shows how we parallelize the method. Results from our implementation using the CM-5 are presented in Section 4. In Section 5, we extend our results to the multi-stage case and show how the same factorization procedure can be used. We also present computational results in this case. We present our conclusions in the last section.

2. Development / Background

Two-Stage Stochastic Linear Programs

The primal form of the linear program can be written as follows:

$$\begin{array}{rlcl} \text{minimize} & c^T x & & \\ \text{subject to} & Ax & = & b \\ & x & \geq & 0 \end{array}$$

where x is a vector of decision variables that are linearly weighted by the elements of c. $Ax = b$ is a set of constraints which the variables must meet along with being non-negative ($x \geq 0$).

Some linear programs involve decisions that are made over a multi-period horizon and are called multi-stage or multi-period linear programs. These problems are created by having decision variables for each period, constraints for each period, and constraints which affect variables in adjoining periods. The constraint matrix of a multi-stage linear program has a staircase structure which is evident in the problem formulation:

$$\begin{array}{lccccccccc} \min & c_1^T x_1 & + & c_2^T x_2 & \cdots & & + & c_P^T x_P & & \\ \text{s.t.} & A_{11} x_1 & & & & & & & = & b_1 \\ & A_{21} x_1 & + & A_{22} x_2 & & & & & = & b_2 \\ & & & & \ddots & & & & & \vdots \\ & & & & & A_{P,P-1} x_{P-1} & + & A_{PP} x_P & = & b_P \\ & & & & & & & x_i & \geq & 0 \end{array}$$

Here P is the number of periods in the model.

We now consider the formulation of two-stage stochastic linear programs. We write the two-stage LP as follows to simplify notation:

$$\begin{array}{lccccc} \min & c^T x & + & d^T y & & \\ \text{s.t.} & A_0 x & & & = & b_0 \\ & Tx & + & Wy & = & b_1 \\ & & & x & \geq & 0 \\ & & & y & \geq & 0 \end{array}$$

Here the first-stage problem data (A_0, c, b_0) are assumed to be known with accuracy. The second-stage problem data (T, W, d, b_1) are uncertain and can be considered as random variables. One method of modelling this uncertainty is to consider the recourse problem.

The recourse process occurs in three stages:

1. The decision maker chooses a value of x, the first-stage decision variable (subject to $A_0 x = b_0$). This decision is made before any future second-period events take place and so cannot anticipate any one future realization over another.

2. A random event occurs determining T, W, d, and b_1.

3. The decision maker chooses a recourse action y, weighted by the cost vector d, which satisfies the constraint $Tx + Wy = b_1$.

The goal is to minimize the sum of the first-stage costs plus the expected second-stage cost of the recourse action. The recourse problem can be written as:

$$\begin{array}{rc} \min_x & c^T x + E\{Q(x,\omega)\} \\ \text{subject to} & A_0 x = b \\ & x \geq 0 \end{array}$$

where $Q(x,\omega)$ is the solution of:

$$\begin{array}{rc} \min_y & d(\omega)^T y \\ \text{subject to} & T(\omega)x + W(\omega)y = b_1(\omega) \\ & y \geq 0 \end{array}$$

Here d, T, W, and b_1 can be thought of as functions of a random variable ω.

The recourse problem can be written as a deterministic linear program if the uncertainty in the problem can be discretized. This discretization can be performed by considering the different possible future outcomes and enumerating them as scenarios. The use of more scenarios will better model the uncertainty and provide a better solution. We will see that more scenarios also requires more solution time. Given N scenarios each with probability p_i of realization, the deterministic equivalent can be written:

$$\begin{array}{lccccccccccc} \min & c^T x & + & p_1 q_1^T y_1 & + & p_2 q_2^T y_2 & \cdots & + & p_N q_N^T y_N^T & & & \\ \text{s.t.} & A_0 x & & & & & & & & & \geq & b \\ & T_1 x & + & W_1 y_1 & & & & & & & = & b_1 \\ & T_2 x & & & + & W_2 y_2 & & & & & = & b_2 \\ & \vdots & & & & & \ddots & & & & & \vdots \\ & T_N x & & & & & & + & W_N y_N & & = & b_N \\ & & & & & & & & x & & \geq & 0 \\ & & & & & & & & y_i & & \geq & 0 \end{array}$$

Notice that there is only one copy of x, the first-stage decision variable. This is the result of the non-anticipativity requirement. On the other hand, note that there is a different vector y_i for each scenario. This is because each second-stage scenario will have a different recourse action. The deterministic equivalent formulation will be used to solve the recourse problem by using an interior-point method.

Interior-Point Method

We use a single-phase, predictor-corrector, primal-dual, path-following variant of an interior-point method. The method is more fully described by Lustig, Marsten, and Shanno [12] and is based on work by Mehrotra [14].

In short, the method chooses an infeasible starting point and iterates through the following steps until the primal and dual infeasibilities and the relative duality gap are within a certain tolerance.

1. Compute primal and dual infeasibilities.
2. Compute the first derivative of the trajectory.
3. Compute the centering parameter.
4. Compute the second derivative.
5. Compute a step length and search direction.
6. Compute step factor.
7. Compute new point.
8. Compute duality gap and objective value.

The most significant portion of the work for performing an iteration of the interior-point method is determining the search direction. This requires the formation of the augmented system

$$\begin{bmatrix} -D^2 & A^T \\ A & 0 \end{bmatrix} \begin{pmatrix} \Delta x \\ \Delta \pi \end{pmatrix} = \begin{pmatrix} v \\ w \end{pmatrix} \tag{2.1}$$

and its factorization. Here A is the constraint matrix from the linear program, D^2 is a diagonal scaling matrix which depends on the current iterate, and v and w are appropriate vectors for the method which depend on the primal and dual infeasibilities of the current solution. The factorization is performed using sparse Bunch-Parlett factorization as described by Fourer and Mehrotra [9]. The algorithm determines a pivot ordering for the matrix based on the sparsity pattern of the matrix and numerical stability of the factors. From one iteration of the method to the next the sparsity pattern of the matrix does not change but the entries in the matrix D^2 do. Even though the pivot ordering from previous iteration is not guaranteed to be stable, we reuse the ordering until we determine that the factor has become unstable by using a simple test. We have noticed that oftentimes it is possible to use one or two pivot ordering to solve the entire linear program.

We choose to operate on the augmented system using a sophisticated indefinite solver rather than forming the normal equations for use with a standard Cholesky solver for a variety of reasons. Because the normal equations involve a matrix-matrix transpose product, any dense column in the matrix causes the resulting system to be completely dense. The columns of the constraint matrix associated with the first-stage variables have entries in rows for each scenario. Although the columns are not completely dense, the resulting system becomes significantly dense. This increase in density causes the solution time to increase cubically as the number of scenarios increases. A more complete discussion of these points can be found in our paper [6].

Motivating Example

To motivate the steps of the factoring procedure that we use, consider the solution of the following system of equations:

$$\begin{bmatrix} E_1 & & F_1^T \\ & E_2 & F_2^T \\ F_1 & F_2 & G \end{bmatrix} \begin{pmatrix} y_1 \\ y_2 \\ x \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ g \end{pmatrix}$$

If E_1 and E_2 are invertible, Gaussian elimination can be performed by multiplying the first row by $-F_1E_1^{-1}$ and adding into the third row and similarly for the second row. This results in the following system:

$$\begin{bmatrix} E_1 & & F_1^T \\ & E_2 & F_2^T \\ 0 & 0 & G - \sum_{i=1}^{2} F_iE_i^{-1}F_i^T \end{bmatrix} \begin{pmatrix} y_1 \\ y_2 \\ x \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ g - \sum_{i=1}^{2} F_iE_i^{-1}f_i \end{pmatrix}$$

which can be rewritten as

$$\begin{bmatrix} E_1 & & F_1^T \\ & E_2 & F_2^T \\ 0 & 0 & G^+ \end{bmatrix} \begin{pmatrix} y_1 \\ y_2 \\ x \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ g^+ \end{pmatrix}$$

where $G^+ = G - \sum_{i=1}^{2} F_iE_i^{-1}F_i^T$ and $g^+ = g - \sum_{i=1}^{2} F_iE_i^{-1}f_i$.

The system

$$G^+x = g^+$$

is solved for x enabling y_1 and y_2 to be computed by solving

$$\begin{aligned} y_1 &= E_1^{-1}(f_1 - F_1^T x) \\ y_2 &= E_2^{-1}(f_2 - F_2^T x) \end{aligned}$$

We will use a similar process to factor the matrix when solving for the search directions.

Factorization

If we form the augmented system of equations for the two-scenario problem, we see that there is special structure of which we can take advantage.

$$\left[\begin{array}{ccc|ccc} D_0^2 & & & A_0^T & T_1^T & T_2^T \\ & D_1^2 & & & W_1^T & \\ & & D_2^2 & & & W_2^T \\ \hline A_0 & & & & & \\ T_1 & W_1 & & & 0 & \\ T_2 & & W_2 & & & \end{array}\right]$$

The matrix can be partitioned by scenarios and reordered to exploit the structure. We will take advantage of the structure in our factorization routine and eventually distribute the factorization across parallel processors. Here is shown the reordered augmented system for a two-scenario problem:

$$\left[\begin{array}{cc|cc|cc} D_1^2 & W_1^T & & & 0 & \\ W_1 & 0 & & & T_1 & \\ \hline & & D_2^2 & W_2^T & 0 & \\ & & W_2 & 0 & T_2 & \\ \hline 0 & T_1^T & 0 & T_2^T & D_0^2 & A_0^T \\ & & & & A_0 & 0 \end{array}\right]$$

Notice that each block on the diagonal is a smaller augmented system.

We can write the complete augmented system in the following compact form:

$$\begin{bmatrix} E_1 & & F_1^T \\ & E_2 & F_2^T \\ F_1 & F_2 & G \end{bmatrix} \begin{pmatrix} y_1 \\ y_2 \\ x \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ g \end{pmatrix}$$

where $E_i = \begin{bmatrix} D_i^2 & W_i^T \\ W_i & 0 \end{bmatrix}$, $F_i = \begin{bmatrix} 0 & T_i^T \\ 0 & 0 \end{bmatrix}$, and $G = \begin{bmatrix} D_0^2 & A_0^T \\ A_0 & 0 \end{bmatrix}$.

In the example in the previous section, we assumed that the matrix E was invertible in order to perform Gaussian elimination. We will be performing similar steps and inverting a matrix of the following form:

$$\begin{bmatrix} D^2 & W^T \\ W & 0 \end{bmatrix}.$$

This system (a subsystem of the complete augmented system) is not guaranteed to be invertible. For the discussion here, we will assume that the system is invertible and consider the singular case in the following section. The sparse Bunch-Parlett factorization method that we use utilizes standard 1×1 pivots along with 2×2 pivots where necessary to maintain numerical stability. This results in a system of the following form:

$$LBL^T$$

where L is a triangular matrix and B is a tridiagonal matrix. It is interesting to note that for some matrices, it is not necessary to use 2×2 pivots and so B is a simple diagonal matrix.

The factorization proceeds as follows:

1. Perform sparse Bunch-Parlett factorization of E_i forming $L_iB_iL_i^T$ for $i = 1, 2$.

2. Update G such that

$$\begin{aligned} G^+ &= G - \textstyle\sum_{i=1}^2 F_i E_i^{-1} F_i^T \\ &= G - \textstyle\sum_{i=1}^2 F_i \left(L_i B_i L_i^T\right)^{-1} F_i^T \\ &= G - \textstyle\sum_{i=1}^2 F_i L^{-T} B_i^{-1} L_i^{-1} F_i^T \\ &= G - \textstyle\sum_{i=1}^2 \left(L^{-1} F_i^T\right)^T B_i^{-1} \left(L^{-1} F_i^T\right) \end{aligned}$$

and update the right-hand side accordingly.

3. Solve the resulting system $G^+x = g^+$.
4. Given x, solve for y_1 and y_2 as follows:

$$y_1 = E_1^{-1}\left(f_1 - F_1^T x\right)$$
$$y_2 = E_2^{-1}\left(f_2 - F_2^T x\right)$$

Because the scenarios of the problem have much in common, it is easy to take advantage of the special structure of these problems. First of all, the zero-nonzero structure of all the T matrices is the same; this is true for the W matrices as well. This similarity extends to the structure of the entire augmented system so that only one copy of the structure of T, W, and the augmented system needs to be stored. It is also possible in many instances to share the factorization pivot ordering among the scenario subproblems. Because the matrices F are such that the matrices on the second row are 0, all updates into G actually only update the D_0^2 submatrix; the matrix A_0 of G is unchanged.

Unstable Factorizations

By partitioning the matrix by scenarios so that it can be factored by blocks, we have imposed a pivot ordering on the system. Even if the entire system is stable, there is no guarantee that each subsystem on the diagonal can be factored. Although the result is not as clean as in the stable case, such unstable factorizations can be handled by our method.

The matrix to be factored is

$$\begin{bmatrix} E_1 & & F_1^T \\ & E_2^T & F_2^T \\ F_1 & F_2 & G \end{bmatrix}$$

where E_i, F_i, and G are defined as before. The first step of the factorization is to decompose E_i into LDL^T where L is lower triangular and D is tridiagonal. If it is not possible to pivot on all the diagonal entries, it may be possible to include those pivots (along with the corresponding rows and columns) with the matrix G yet to be factored.

More specifically, consider the system

$$\begin{bmatrix} E & F^T \\ F & G \end{bmatrix}$$

where E cannot be completely factored. Since part of the matrix E can be factored, we will implicitly permute the matrix and call that part E_{11}. The resulting system has the following form:

$$\begin{bmatrix} E_{11} & E_{21}^T & F_1^T \\ E_{21} & E_{22}^T & F_2^T \\ F_1 & F_2 & G \end{bmatrix}$$

Here we have partitioned the matrix F to correspond with the reordering and partitioning of E. This is another augmented system if we consider

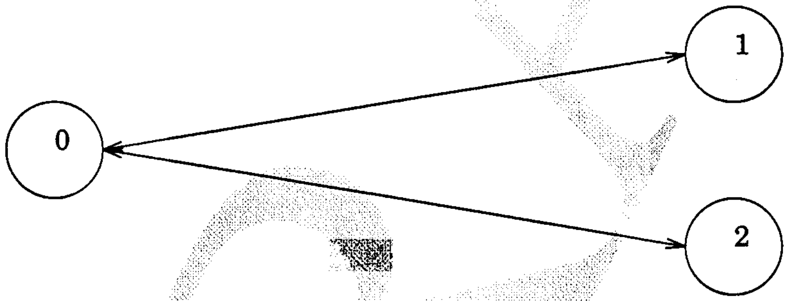
$$\begin{bmatrix} \tilde{E} & \tilde{F}^T \\ \tilde{F} & \tilde{G} \end{bmatrix}$$

where $\tilde{E} = E_{11}$, $\tilde{F} = \begin{bmatrix} E_{21} \\ F_1 \end{bmatrix}$, and $\tilde{G} = \begin{bmatrix} E_{22} & F_2^T \\ F_2 & G \end{bmatrix}$. In essence, the unstable portion of the matrix E (along with rows of F and columns of F^T) are appended to the original G matrix. If there is more than one unstable matrix E_i on the diagonal, the matrix $\tilde{G}$ may include additional rows and columns from any or all of the E matrices.

3. Parallel Implementation

The Gaussian elimination sample problem in the previous section can be easily parallelized. By storing E_1, F_1, and f_1 on processor 1 and E_2, F_2, and f_2 on processor 2, the computation of $F_i E_i^{-1} F_i^T$ and $F_i E_i^{-1} f_i$ can be performed concurrently. A third processor stores G and g and receives the update matrices and right-hand side as messages from the first two processors. The third processor computes the value of x and sends it to the other processors which then compute y_1 and y_2. Computing the solution of x on the third processor is inherently serial, and the other processors are forced to remain idle until x has been computed.

Figure 3–1 shows the step-by-step procedure with a schematic diagram. The circles represent the processors and associated data. The lines with arrows represent communication between processors.



	Compute E_i on each processor. Compute $F_i A_i^{-1} F_i^T$ and $F_i A_i^{-1} f_i$ on each processor. Send matrix and vector to Processor 0.
Receive matrices and vectors from other processors. Subtract matrices from G. Subtract vectors from g. Solve $Gx = g$. Send x to other processors.	
	Receive x from Processor 0. Compute $y_i = E_i^{-1}(f_i - F_i^T x)$.

Figure 3–1. *Processor schematic* for example problem

In the same way, the factorization of the augmented system for the stochastic linear program can be partitioned across the processors. In fact, it is possible to distribute all the computation for the problem across processors. Matrix and vector addition, matrix-vector products, vector dot products, the minimum-ratio test, and all the other operations can be performed in parallel. How we perform these operations depends on how the problem is laid out.

We seek to lay the problem out so that the second-stage scenarios are distributed evenly over the processors. For ease of discussion, we assume that the number of scenarios is a multiple of the number of processors used to solve the problem. If there are 3,200 scenarios, then each processor will perform computations for 100 scenarios on a 32-processor machine. In addition, one processor (processor 0) will perform the computation for the first-stage along with the computation for its allocation of the second-stage scenarios.

Each processor reads in the entire multi-scenario problem. After determining for which scenarios each processor will compute, only the data for those scenarios is retained. In the 3,200-scenario case, each processor stores information for each of its 100 scenarios. Processor 0 will also store the A_0 matrix and other information for the first-stage. The information that each processor has includes the matrices T and W, the right-hand side b, the cost vector c, and the various other vectors that are needed by the method. All of the data pertaining to a scenario remains local to the processor.

Each processor runs the same program although the processors execute different instructions. The program is written so that a processor can use its processor number as a test to determine which portions of the code should be executed. The factorization process proceeds similarly to the example above. Every processor (including 0) executes the factorization of E_i, the matrix update $(L^{-1}F^T)B_i^{-1}(L^{-1}F^T)^T$, and the vector update. The two updates are sent to Processor 0 via message-passing routines. Processor 0 receives the updates from all of the processors and computes the solution x of the system $G^+x = g^+$. Processor 0 sends x to the other processors which have been idle waiting for the solution from Processor 0. All of the computation in the interior-point method occurs in this manner. Processor 0 sends and receives messages from all the other processors. The other processors do not explicitly communicate with one another. Notice that since each processor is performing the computation for more than one scenario, an efficient implementation can take advantage of the common structure of the different matrices and a common factorization routine.

Since communication is an expensive operation in a parallel computer, reducing communication can greatly speed up a routine. The algorithm as described above has Processor 0 receiving update matrices and vectors from each different scenario. Since many different scenarios reside on a processor, it makes sense to have each processor accumulate the updates for its set of scenarios and then send one update to Processor 0. This can be shown schematically as in Figure 3–2.

The dashed circles represents the accumulation of matrices and vectors within the processor. The dashed lines represent "communication" within the processor to perform the accumulation. As a result, the amount of communication required for the method is a function of the number of processors rather than the number of scenarios in the problem.

4. Computational Results

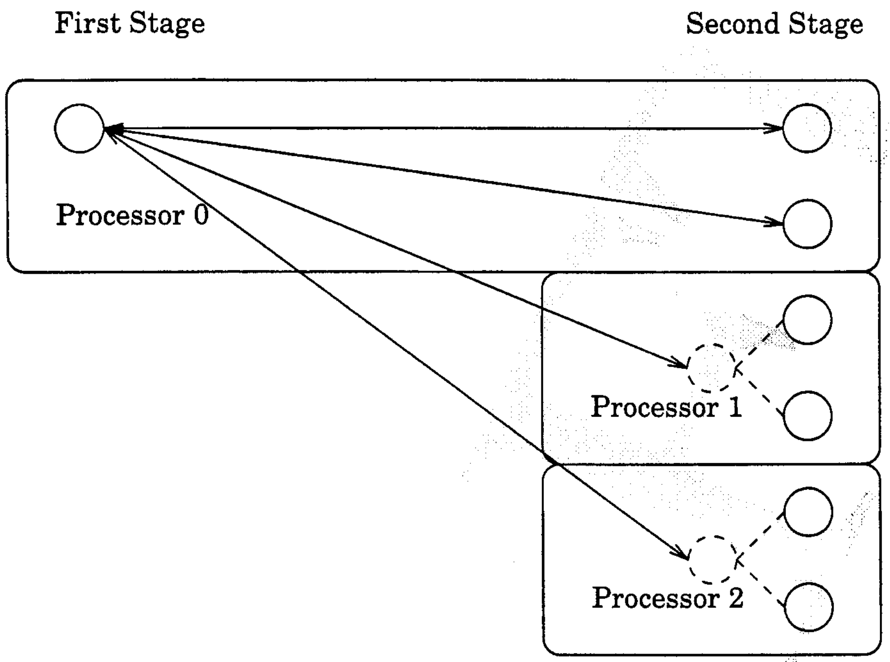


Figure 3–2. *Schematic showing processor communication* for solving two-stage stochastic linear programs

Our program was written in C and utilized CMMD (the Connection Machine's message passing library) routines for all inter-processor communication. The programs were run on NCSA's CM-5 in Champaign, Illinois using 32, 64, or 128 processors.

We report results for solving problems with varying numbers of scenarios. The problem SCSD8 is taken from the NETLIB [10] testset and made a stochastic linear program by considering different elements of the right-hand side vector to be random. We created random problems with between 32 and 2560 scenarios.

The original problem, SCSD8, has 70 variables and 11 constraints in the first period. There are 140 variables and 20 constraints in the second. Since the second-stage variables and constraints are duplicated for each scenario included in the problem, the number of rows and columns increases with the number of scenarios. Table 4–1 shows the total number of rows and columns in the problems that we solved.

Note: The factorization routine that has been used is not a complete implementation of the sparse Bunch-Parlett factorization. The current preliminary routine merely performs 1×1 *pivots along the diagonal. When an unstable pivot occurs, the program merely stops execution and prints the current result. Although this factorization routine is not robust, many of the stochastic linear programs were solved to optimality. For one problem, SCSD8, all of the problems generated were solved to optimality. Although the following results cannot be used to comment on the robustness of the solutions, these results do show that the problems can be decomposed for factorization and that parallel speedups are possible.*

In Figure 4–3 we have plotted the time per iteration against the number of scenarios per processor. Note that the solution time grows linearly with the number

Scenarios	Rows	Columns
32	651	4550
64	1291	9030
128	2571	17990
160	3211	22470
320	6411	44870
640	12811	89670
1280	25611	179270
1600	32011	224070
1920	38411	268870
2560	51211	358470

Table 4–1. *SCSD8:* Problem dimensions for two-stage problems

of scenarios per processor. The use of the augmented system results in such linear growth. Had the normal equations approach been used, the solution time would have grown cubically in the number of scenarios. Note also that the lines for 32, 64, and 128 scenarios are parallel; each individual processor does the same amount of work whether we are using a 32, 64, or 128 processor machine. The intercept of the plots is different for each size of machine since there is more communication overhead for larger machines. The difference in the y-intercept is a measure of the communication overhead.

The solution time per iteration is composed of three distinct components:

1. Computation time for second-period scenarios,
2. Communication time from all processors to Processor 0, and
3. Computation time for first-period solves and other overhead.

Let:

T	$=$	Total solution time per iteration
s_2	$=$	Number of second-stage scenarios
p	$=$	Number of processors

Then s_2/p is the number of second-stage scenarios per processor. The total solution time per processor can then be written

$$T = a\frac{s_2}{p} + bp + c$$

where a is the time for each processor to perform the computation for one of its scenarios, b is the amount of time for Processor 0 to communicate with another processor, and c is the amount of time for Processor 0 to perform the necessary first-stage computations serially.

Using simple linear regression, it is easy to determine the values for the constants a, b, and c.

a	b	c
0.450	0.0336	0.207

and

$$T = 0.450\frac{s_2}{p} + 0.0336p + 0.207$$

Scenarios	Iterations	Objective Value	Rel. Dual Gap	Primal Inf.	Dual Inf.	Time (secs)
32 processors						
32	9	21.08065	1.45e-13	1.83e-10	1.33e-16	15.35
160	11	20.92091	3.95e-14	2.73e-10	1.44e-16	38.01
320	13	20.90149	4.48e-13	3.55e-13	2.10e-16	74.20
640	12	20.89766	2.17e-10	5.49e-10	2.01e-16	123.12
1280	13	20.88141	1.10e-11	1.25e-13	2.75e-16	251.88
1600	13	20.88618	4.72e-10	5.49e-11	7.83e-16	314.21
2560	14	20.90191	3.56e-11	1.76e-12	3.08e-16	518.50
64 processors						
64	9	20.97917	8.07e-14	9.45e-10	2.01e-16	25.92
320	13	20.90149	8.89e-14	3.55e-13	1.89e-16	60.20
640	12	20.89766	2.17e-10	5.49e-10	2.12e-16	82.42
1280	13	20.88141	1.66e-12	1.25e-13	1.97e-16	145.96
2560	14	20.90191	7.79e-15	1.76e-12	3.47e-16	288.75
128 processors						
128	10	20.92922	5.01e-11	3.98e-08	3.30e-16	50.04
640	12	20.89766	2.17e-10	5.49e-10	3.43e-16	80.82
1280	13	20.88141	1.04e-11	1.25e-13	2.67e-16	116.34
1920	13	20.89127	4.82e-14	1.16e-11	2.61e-16	146.90
2560	14	20.90191	4.54e-15	1.76e-12	3.48e-16	187.60

Table 4–2. *Solution times* for two-stage problems

$R^2 = 0.999$ for this model so this equation is an excellent fit to the data.

If

$$0.450\frac{s_2}{p} \gg 0.0336p$$

or

$$s_2 \gg \frac{p^2}{13.39},$$

the first term (second-stage computation time) dominates the solution time. We see then that the solution time will approximately double as the number of scenarios in the problem doubles. Also, as long as the above condition is met, doubling the number of processors will halve the solution time.

5. Extensions for the Multi-stage Case

Our solution method for two-stage stochastic linear programs extends nicely to the multi-stage case. We will first introduce multi-stage problems and then describe the changes needed in the factorization method.

We will consider a three-stage problem since considering additional stages is

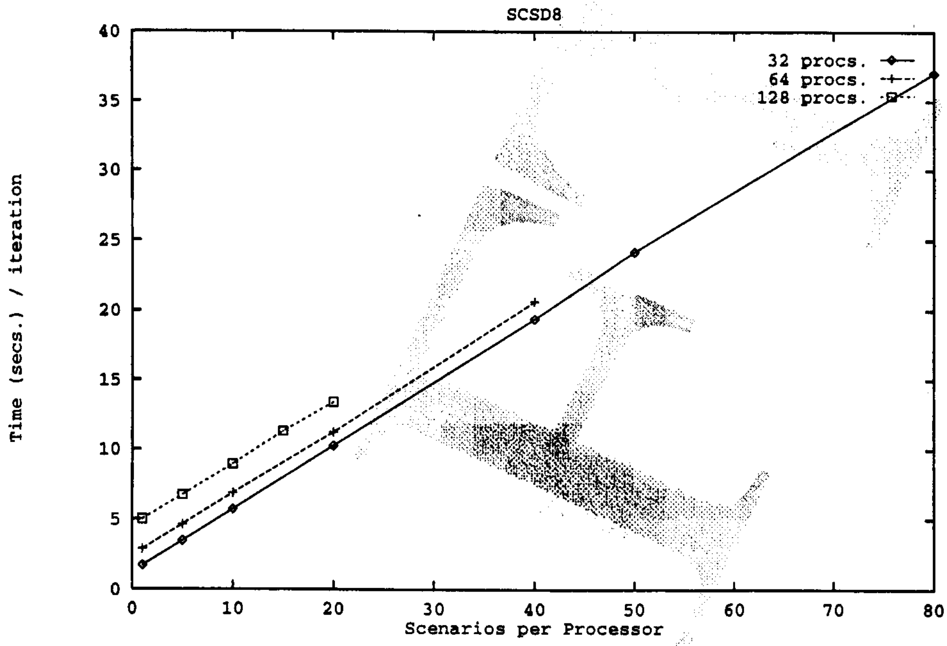


Figure 4–3. *Solution times* for two-stage problems

trivial. We first consider the three-stage linear program:

$$\begin{array}{rccccccl}
\min & c^T x & + & d_2 y_2 & + & d_3 y_3 & & \\
\text{subject to} & A_0 x & & & & & = & b_0 \\
 & T_1 x & + & W_1 y_2 & & & = & b_1 \\
 & & & T_2 y_2 & + & W_2 y_3 & = & b_2 \\
 & & & & & x & \geq & 0 \\
 & & & & & y_i & \geq & 0
\end{array}$$

Multi-stage stochastic linear programs are similar to the two-stage problems in that we are trying to account for uncertainty in the problem data. The two-stage case assumes that all future information will be revealed at one point in the future (step 2 of the recourse process). In contrast, the multi-stage process has the future being revealed at separate points in time and allows for recourse actions at those times. The process occurs as follows:

1. The decision maker chooses a value of x, the first-stage decision variable (subject to $A_0 x = b_0$). This decision is made before any future second-period events take place and so cannot anticipate any one future realization over another.

2. A random event occurs determining T_1, W_1, d_2 and b_1.

3. The decision maker chooses a recourse actions y_2 weighted by the cost vector d_2, satisfying the constraint $T_1 x + W_1 y_2 = b_1$.

4. A random event occurs determining T_2, W_2, d_3 and b_2.

5. The decision maker chooses a recourse actions y_3 weighted by the cost vector d_3, satisfying the constraint $T_2 y_2 + W_2 y_3 = b_2$.

As in the two-stage case, we can write the deterministic equivalent. This is a three-stage problem with two scenarios in both the second and third stage.

$$\begin{array}{lllllllll}
\min & c^T x + p_{21} d_{21}^T y_{21} + p_{22} d_{22}^T y_{22} + p_{31} d_{31}^T (y_{31} + y_{33}) + p_{32} d_{32}^T (y_{32} + y_{34}) & & & & & & & \\
\text{st} & A_0 x & & & & & & = & b_0 \\
 & T_{11} x \; + \; W_{11} y_{21} & & & & & & = & b_{11} \\
 & \qquad T_{21} y_{21} \; + & W_{21} y_{31} & & & & & = & b_{21} \\
 & \qquad T_{22} y_{21} \; + & & W_{22} y_{32} & & & & = & b_{22} \\
 & T_{12} x \; + & & & W_{12} y_{22} & & & = & b_{12} \\
 & & & & T_{21} y_{22} \; + & W_{21} y_{33} & & = & b_{21} \\
 & & & & T_{22} y_{22} \; + & & W_{22} y_{34} & = & b_{22}
\end{array}$$

Notice that there is one copy of x since the first-stage decision cannot anticipate any one future realization over another. There are two 2nd-stage variables, y_{21} and y_{22}, since this decision is made after T_{11}, W_{11}, b_{11} or T_{12}, W_{12}, b_{12} have been revealed. This matrix has an arborescent structure which we will exploit.

We form the augmented system as in the two-stage case and can reorder it for factorization. The resulting system has the following structure:

$$\left[\begin{array}{cccccc|cccccc|cc}
D_{31}^2 & W_{21}^T & & & & & & & & & & & & \\
W_{21} & 0 & & & T_{21} & & & & & & & & & \\
 & & D_{32}^T & W_{22}^T & & & & & & & & & & \\
 & & W_{22} & 0 & T_{22} & & & & & & & & & \\
 & T_{21}^T & & T_{22}^T & D_{21}^2 & W_{11}^T & & & & & & & & \\
 & & & & W_{11} & 0 & & & & & & & T_{11} & \\
\hline
 & & & & & & D_{33}^2 & W_{21}^T & & & & & & \\
 & & & & & & W_{21} & 0 & & & T_{21} & & & \\
 & & & & & & & & D_{34}^T & W_{22}^T & & & & \\
 & & & & & & & & W_{22} & 0 & T_{22} & & & \\
 & & & & & & & T_{21}^T & & T_{22}^T & D_{22}^2 & W_{11}^T & & \\
 & & & & & & & & & & W_{11} & 0 & T_{12} & \\
\hline
 & & & & & T_{11}^T & & & & & & T_{12}^T & D_{11} & A_0^T \\
 & & & & & & & & & & & & A_0 & 0
\end{array}\right]$$

The matrix has been partitioned so that there are two complete two-stage augmented systems on the diagonal. That is, there are two smaller augmented systems within the larger system and within those two are two smaller systems each.

Let us write the above system as

$$\left[\begin{array}{ccc|ccc|c}
E_{11} & & F_{11}^T & & & & \\
 & E_{12} & F_{12}^T & & & & \\
F_{11} & F_{12} & G_1 & & & & \\
\hline
 & & & E_{21} & & F_{21}^T & \\
 & & & & E_{22} & F_{22}^T & \\
 & & & F_{21} & F_{22} & G_2 & \\
\hline
 & & F_1 & & & F_2 & G
\end{array}\right]
\begin{pmatrix} y_{11} \\ y_{12} \\ x_1 \\ y_{21} \\ y_{22} \\ x_2 \\ x \end{pmatrix}
=
\begin{pmatrix} f_{11} \\ f_{12} \\ g_1 \\ f_{21} \\ f_{22} \\ g_2 \\ g \end{pmatrix}$$

or

$$\begin{bmatrix} E_1 & & F_1^T \\ & E_2 & F_2^T \\ F_1 & F_2 & G \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ g \end{pmatrix}$$

Here we see that adding more stages to the problem merely nests augmented systems within other augmented systems. The more stages that you consider, the more nesting that results.

The factorization process is identical on a macro level. The matrices E_i are factored and updates of $(L_i^{-1}F_i)^T B_i^{-1}(L_i^{-1}F_i)$ are subtracted from G and similarly for g. The vector x is determined by solving $G^+x = g^+$. Given x, the vectors y_1 and y_2 can be easily determined. Although these steps remain the same, the process of factoring E_i is more complicated. In the two-stage case, E_i has a simple augmented structure $\begin{bmatrix} D^2 & W^T \\ W & 0 \end{bmatrix}$; however, in the three-stage case, it has an entire two-stage structure embedded within it $\begin{bmatrix} E_1 & & F_1^T \\ & E_2 & F_2^T \\ F_1 & F_2 & G \end{bmatrix}$. As a result, factoring E_i is to complete the factorization process for the two-stage case.

The entire factorization process happens as follows:

1. Factor the augmented systems E_{11}, E_{12}, E_{21}, and E_{22}.

2. Given the factors of E_{11} and E_{12}, update:

$$\begin{aligned} G_1^+ &= G_1 - F_{11}E_{11}^{-1}F_{11}^T - F_{12}E_{12}^{-1}F_{12}^T \\ &= G_1 - (L_{11}^{-1}F_{11})^T B_{11}^{-1}(L_{11}^{-1}F_{11}) - (L_{12}^{-1}F_{12})^T B_{12}^{-1}(L_{12}^{-1}F_{12}) \end{aligned}$$

and

$$\begin{aligned} G_2^+ &= G_2 - F_{21}E_{21}^{-1}F_{21}^T - F_{22}E_{22}^{-1}F_{22}^T \\ &= G_2 - (L_{21}^{-1}F_{21})^T B_{21}^{-1}(L_{21}^{-1}F_{21}) - (L_{22}^{-1}F_{22})^T B_{22}^{-1}(L_{22}^{-1}F_{22}) \end{aligned}$$

We have now collapsed the third stage into the second, and the resulting system now resembles a two-stage problem.

3. Factor G_1^+ into $L_1B_1L_1^T$ and G_2^+ into $L_2B_2L_2^T$.

4. Update:

$$\begin{aligned} G^+ &= G - F_1G_1^{-1}F_1^T - F_2G_2^{-1}F_2^T \\ &= G - (L_1^{-1}F_1)^T B_1^{-1}(L_1^{-1}F_1) - (L_2^{-1}F_2)^T B_2^{-1}(L_2^{-1}F_2) \end{aligned}$$

5. Solve $G^+x = g^+$ for x.

6. Given x, compute:

$$\begin{aligned} x_1 &= (G_1^+)^{-1}(g_1^+ - F_1^T x) \\ x_2 &= (G_2^+)^{-1}(g_2^+ - F_2^T x) \end{aligned}$$

7. Given x_1 and x_2, compute

$$\begin{aligned} y_{11} &= E_{11}^{-1}(f_{11} - F_{11}^T x_1) \\ y_{12} &= E_{12}^{-1}(f_{12} - F_{12}^T x_1) \\ y_{21} &= E_{21}^{-1}(f_{21} - F_{21}^T x_1) \\ y_{22} &= E_{22}^{-1}(f_{22} - F_{22}^T x_1) \end{aligned}$$

Notice that this process can be partitioned across processors just as with the two-stage problem. Figure 5–1 shows this schematically. In order to evenly divide the scenarios across the processors, we take into account the number of second- and third-stage scenarios and distribute the scenarios so that each processor performs approximately equal amounts of computation. Notice that there is no additional inter-processor communication; communication occurs between Processor 0 only for steps 4, 5, and 6 in order to compute the first-stage variables.

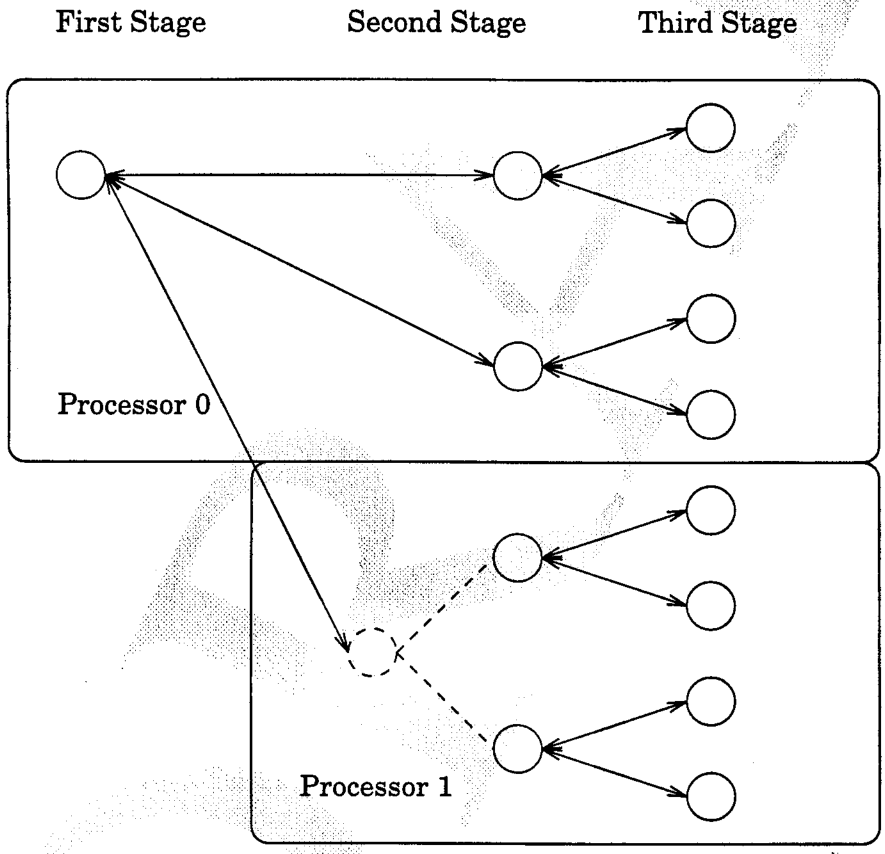


Figure 5–1. *Factorization process* for three-stage problems

Table 5–2 shows the results for solving various three-stage problems. The number of second-stage scenarios varies between 32, 160, and 320 on a 32-processor machine and 64 and 640 on a 64-processor machine. The number of third-stage scenarios varies among 1, 10, 100, and 500 scenarios. The largest problem includes 320 second-stage scenarios and a total of 32,000 third-stage scenarios. Table 5–3 shows the total number of rows and columns in the problems that we solved. Figure 5–4 plots these results. The results for the 32-processor machine are on the left, and the 64-processor machine on the right.

2nd-stage Scenarios	3rd-stage Scenarios	Total Scens	Iters	Objective Value	Rel. Dual Gap	Primal Inf.	Dual Inf.	Time (secs)
32 processors								
32	1	32	9	21.080650	5.84e-13	1.83e-10	1.64e-16	25.80
32	10	320	9	24.347310	1.07e-13	2.77e-10	1.07e-10	40.36
32	100	3200	11	24.052770	4.81e-12	3.52e-12	4.17e-11	234.50
32	500	16000	12	24.027960	6.70e-12	2.14e-11	1.12e-11	1140.24
160	1	160	11	20.920910	3.10e-14	2.73e-10	1.92e-16	94.58
160	10	1600	10	24.187580	7.68e-12	2.97e-09	1.16e-16	169.79
160	100	16000	11	23.893030	2.10e-11	8.31e-10	7.78e-12	1112.34
320	1	320	13	20.901490	5.40e-13	3.55e-13	1.90e-16	206.37
320	10	3200	11	24.168160	6.03e-09	8.65e-09	2.14e-16	364.09
64 processors								
64	1	64	9	20.979170	2.59e-14	9.45e-10	1.97e-16	35.15
64	10	640	9	24.245830	8.26e-09	1.54e-07	7.69e-09	51.03
64	100	6400	11	23.951290	1.45e-12	2.61e-12	4.35e-12	242.04
64	500	32000	12	23.926480	6.30e-12	1.49e-11	5.01e-12	1143.17
320	1	320	13	20.901490	9.87e-13	3.55e-13	2.27e-16	126.80
320	10	3200	11	24.168160	6.03e-09	8.65e-09	1.77e-16	204.26
320	100	32000	13	23.873610	2.90e-12	1.01e-12	2.61e-16	1299.20
640	1	640	12	20.897660	2.17e-10	5.49e-10	2.29e-16	207.38
640	10	6400	11	24.165370	8.48e-10	3.52e-09	2.05e-16	385.53

Table 5–2. *Solution times* for three-stage problems

2nd-Stage Scenarios	3rd-Stage Scenarios	Rows	Columns
32	1	650	4550
32	10	3530	24710
32	100	32330	226310
32	500	160330	1122310
64	1	1290	9030
64	10	7050	49350
64	100	64650	452550
64	500	320650	2244550
160	1	3210	22470
160	10	17610	123270
160	100	161610	1131270
320	1	6410	44870
320	10	35210	246470
320	100	323210	2262470
640	1	12810	89670
640	10	70410	492870

Table 5–3. *Problem dimensions* for SCSD8 - 3 stages

We can again use simple linear regression to analyze the time to compute one iteration of the interior-point method. The solution time is now composed of four distinct components:

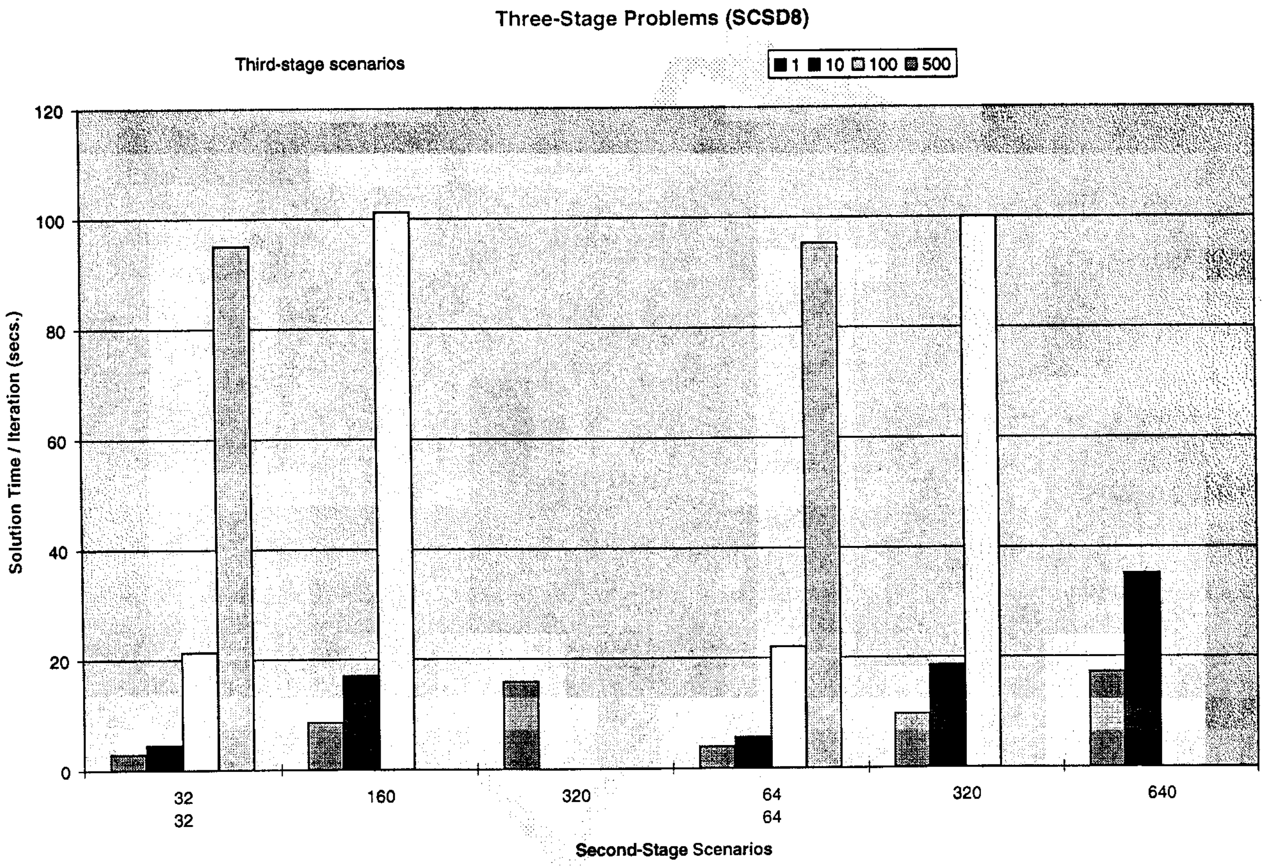


Figure 5–4. *Solution times* for three-stage problems

1. Computation time for second-period scenarios,
2. Computation time for third-period scenarios,
3. Communication time from all processors to Processor 0, and
4. Computation time for first-period solves and other overhead.

Let:

T	=	Total solution time per iteration
s_2	=	Number of second-stage scenarios
s_3	=	Number of third-stage scenarios per second-stage scenario
p	=	Number of processors

Then s_2/p is the number of second-stage scenarios per processor, and $s_2 s_3/p$ is the total number of third-stage scenarios per processor.

The total solution time per processor can then be written

$$T = a\frac{s_2}{p} + b\frac{s_2 s_3}{p} + cp + d$$

where a is the time for each processor to perform the computation for one of its second-stage scenarios, b is the time to perform the computation for a third-stage scenario, c is the amount of time for Processor 0 to communicate with another processor, and d is the amount of time for Processor 0 to perform the necessary first-stage computations serially.

Given the data in Table 5–2, these coefficients can easily be determined.

a	b	c	d
1.33	0.184	0.027	0.489

and

$$T = 1.33\frac{s_2}{p} + 0.184\frac{s_2 s_3}{p} + 0.027p + 0.489$$

$R^2 = 0.999$ for this model so this equation too provides an excellent fit to the data. If

$$1.33\frac{s_2}{p} + 0.184\frac{s_2 s_3}{p} \gg 0.027p$$

or

$$s_2(49.25 + 6.81 s_3) \gg p^2,$$

then the computation time for the scenarios dominates the communication time. If the computational time dominates, then doubling s_2 will double the solution time. Also if $6.81 s_3 \gg 49.25$, doubling s_3 will approximately double the solution time. Similarly, doubling the number of processors will approximately halve the solution time.

6. Conclusions

In this paper, we have described how the structure of a multi-stage stochastic linear program can be exploited by using an interior-point algorithm on a parallel computer. We have successfully solved two-stage problems containing up to 2,560 scenarios (51,211 rows and 358,470 columns) and three-stage problems containing up to 640 second-stage scenarios and 100 third-stage scenarios (the largest being 320 second-stage scenarios and 100 third-stage scenarios with 323,210 rows and 2,262,470 columns) within reasonable amounts of time.

Our routine works well with increasing numbers of scenarios; in fact, the efficiency of the algorithm increases with the number of scenarios per processor. This is important since one can better model random distributions of the problem data using more scenarios. The algorithm handles multiple time periods in the model. This is important since many problems plan over a multi-year time horizon. The ability to plan over many periods leads to more robust solutions. Finally the algorithm works with additional processors so that it can take advantage of more powerful machines as they may become available.

References

[1] J. R. BIRGE, 1985. Decomposition and Partitioning Methods for Stochastic Linear Programs. *Operations Research* **33**, 989–1007.

[2] J. R. BIRGE, M. A. H. DEMPSTER, H. I. GASSMANN, E. A. GUNN, A. J. KING and S. W. WALLACE, 1987. A Standard Input Format for Multiperiod Stochastic Linear Programs. *Committee on Algorithms Newsletter of the Mathematical Programming Society* **17**, 1–19.

[3] J. R. BIRGE, C. J. DONOHUE, D. F. HOLMES, and O. G. SVINTSITSKI, 1994. A Parallel Implementation of the Nested Decomposition Algorithm for Multistage Stochastic Linear Programs. Technical Report 94-1, Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor, MI.

[4] J. R. BIRGE and D. F. HOLMES, 1992. Efficient Solution of Two-Stage Stochastic Linear Programs Using Interior-Point Methods. *Computational Optimization and Applications* **1**, 245–276.

[5] J. R. BIRGE and L. QI, 1988. Computing Block-Angular Karmarkar Projections with Applications to Stochastic Programming. *Management Science* **34**, 1472–1479.

[6] J. CZYZYK, R. FOURER and S. MEHROTRA, 1994. A Study of the Augmented System and Column-Splitting Approaches for Solving Two-Stage Stochastic Linear Programs by Interior-Point Methods. Technical Report 93-05, Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, IL; to appear in *ORSA Journal on Computing*.

[7] A. DE SILVA and D. ABRAMSON, 1994. Tech Report: CIT-94-4, School of Computing and Information Technology, Griffith University, Nathan QLD 4111 Australia.

[8] YU. ERMOLIEV and R. J.-B. WETS, eds., 1988. *Numerical Techniques for Stochastic Optimization*. Springer-Verlag, New York.

[9] R. FOURER and S. MEHROTRA, 1993. Solving Symmetric Indefinite Systems in an Interior-Point Method for Linear Programming. *Mathematical Programming* **62**, 15–39.

[10] D.M. GAY, 1985. Electronic Mail Distribution of Linear Programming Test Problems. *Committee on Algorithms Newsletter* **13**, 10–12. Also Numerical Analysis Manuscript 86-0, AT&T Bell Laboratories, Murray Hill, NJ (1986).

[11] E. R. JESSUP, D. YANG, and S. A. ZENIOS, 1994. Parallel Factorization of Structured Matrices Arising in Stochastic Programming. To appear in *SIAM Journal on Optimization*.

[12] I. J. LUSTIG, R.E. MARSTEN and D. F. SHANNO, 1992. On Implementing Mehrotra's Predictor-Corrector Interior Point Method for Linear Programming. *SIAM Journal on Optimization* **2**, 435–449.

[13] I. J. LUSTIG, J.M. MULVEY and T. J. CARPENTER, 1991. Formulating Two-Stage Stochastic Programs for Interior Point Methods. *Operations Research* **39**, 757–770.

[14] S. MEHROTRA, 1992. On the Implementation of a Primal-Dual Interior Point Method. *SIAM Journal on Optimization* **2**, 575–601.

[15] R. J. VANDERBEI and T. J. CARPENTER, 1993. Symmetric Indefinite Systems for Interior-Point Methods. *Mathematical Programming* **58**, 1–32.

[16] R. M. VAN SLYKE and R. J.-B. WETS, 1969. L-Shaped Linear Programs with Applications to Optimal Control and Stochastic Programming. *SIAM J. App. Math* **17**, 638–663.

DRAFT